pressure of the finger 8 or other conductive object relative to the sensing plane 10.

Detailed Description Text (12):
Besides simple X and Y position information, the sensor technology of the present invention also provides finger pressure information. This additional dimension of information may be used by programs to control special features such as "brush-width modes in Paint programs, special menu accesses, etc., allowing provision of a more natural sensory input to computers. It has also been found useful for implementing "mouse click and drag" modes and for simple input gestures.

Detailed Description Text (21):
According to the preferred embodiment of the invention, a position sensor system including touch sensor array 22 and associated position detection circuitry will detect a finger position on a matrix of printed circuit board traces via the capacitive effect of finger proximity to the touch sensor array 22. The position sensor system will report the X, Y position of a finger placed near the touch sensor array 22 to much finer resolution than the spacing between the first and second sets of conductive traces 26 and 30. The position sensor according to this embodiment of the invention will also report a z value proportional to the outline of that finger and hence indicative of the pressure with which the finger contacts the surface of insulating layer 36 over the touch sensor array 22.

Detailed Description Text (32):
The present invention overcomes this problem by "taking a snapshot" of all inputs simultaneously in X and then Y directions (or visa versa). Because the injected noise is proportional to the finger signal strength across all inputs, it is therefore symmetric around the finger centroid. Because it is symmetric around the finger centroid it does not affect the finger position. Additionally, the charge amplifier performs a, differential measuring function to further reject common-mode noise.

Detailed Description Text (54):
Note that proper sizing of N-channel MOS transistors 102 and 108 may provide temperature compensation. This is accomplished by taking advantage of the fact that the threshold of N-Channel MOS transistor 108 reduces with temperature while the mobility of both N-Channel MOS transistors 102 and 108 reduce with temperature. The threshold reduction has the effect of increasing the current while the mobility reduction has the effect of decreasing the current. By proper device sizing these effects can cancel each other out over a significant part of the operating range.

Detailed Description Text (95):
The finger pressure is obtained by summing the capacitances measured on the sense lines. This value is already present in the denominator accumulator after subtracting the offset O.sub.D. A finger is present if the pressure exceeds a suitable threshold value. This threshold may be chosen experimentally and is a function of surface material and circuit timing. The threshold may be adjusted to suit the tastes of the individual user.

Detailed Description Text (96):
The pressure reported by the device is a simple function f(X.sub.D, Y.sub.D) of the denominators for the X and Y directions as implemented in block 178 of FIG. 8. Possible functions include choosing one preferred denominator value, or summing the denominators. In a presently preferred embodiment, the smaller of the two denominators is chosen. This choice has the desirable effect of causing the pressure to go below the threshold if the finger moves slightly off the edge of the pad, where the X sensors are producing valid data, but the Y sensors are not, or vice versa. This acts as an electronic bezel which can take the place of a mechanical bezel at the periphery of the sensor area.

Detailed Description Text (104):
Subtractor unit 194 is a simple subtractor that computes the signed difference between Z and O.sub.Z, This subtractor is actually redundant with subtractor 172 in FIG. 8, and so may be merged with it in the actual implementation. The output C.sub.Z.of this subtractor is the calibrated Z value, an estimate of the finger pressure. This pressure value is compared against a positive and negative Z threshold by comparators 196 and 198. These thresholds are shown as Z.sub.TH and -Z.sub.TH, although they are

not actually required to be equal in magnitude.

Detailed Description Text (105):
If pressure signal C.sub.Z is greater than Z.sub.TH, the signal FINGER is asserted indicating the possible presence of a finger. The Z.sub.TH threshold used by the calibration unit 180 is similar to that used by the rest of the system to detect the presence of the finger, or it may have a different value. In. the present embodiment, the calibration Z.sub.TH is set somewhat lower than the main Z.sub.TH to ensure that the calibration unit 180 makes a conservative choice about the presence of a finger.

Detailed Description Text (125):
According to this aspect of the present invention, finger motions in the inner zone 240 are translated in the standard fashion into motion events to be sent to the host computer. As is well understood in the art, the standard way to communicate mouse motion to a host computer may also be employed in the present invention to communicate finger motion to a host computer. After the finger position is established as disclosed herein, the information communicated to the host computer is:

Detailed Description Text (127):
where .DELTA.X is the change in the X position of the finger, .DELTA.Y is the change in the Y position of the finger, X.sub.cur is the current X position of the finger and X.sub.old is the last reported X position of the finger, Y.sub.cur is the current Y position of the finger and Y.sub.old is the last reported Y position of the finger, and A is a "gain factor" which is commonly encountered in mouse cursor control applications.

Detailed Description Text (128):
Typically, the host computer takes (.DELTA.X, .DELTA.Y) events and moves the cursor by the indicated amount in each axis, thus reconstructing the finger position on the screen as the successive .DELTA.X and .DELTA.Y values are accumulated. So far, this is standard cursor control behavior where edge motion is not considered.

Detailed Description Text (135):
For example, if the finger is held a good distance to the right (so that X.sub.cur >X.sub.0), then the cursor will tend to "glide" to the right at a constant speed set by multiplicative speed factor S in Eqs. 12 and 13. This factor can be adjusted to individual taste of a user.

Detailed Description Text (140):
As discussed above, when a finger is in the outer zone, the FingerOuter signal, which is a global signal for both the X and Y axis, is made true, and increments are added to the (.DELTA.X, .DELTA.Y) events pursuant to Eqs. 12 and 13. Because increments corresponding to S(X.sub.cur -X.sub.center) and S(Y.sub.cur -Y.sub.center) are added in the X and Y directions, respectively, for the (.DELTA.X, .DELTA.Y) events, the direction of the cursor motion will be along a vector from, the center of the pad to the finger position. In graphical environments, there are many vertical and horizontal objects and use of edge motion may incur some unintended results. For example, if a user in a graphical environment pulls down a tall pop-up menu, the user may need the assistance of the edge motion feature to reach the bottom of the menu. In this case, however, the direction of the cursor motion may cause the cursor to slide off the pop-up menu, when the user actually wants the cursor to move in a vertical motion along the Y axis.

Detailed Description Text (191):
Some number of physical switches may be supported by gesture unit 20. In the illustrative example of FIG. 14, two inputs A and B to button control unit 286 come from physical switches. Such switches may be mounted on the touchpad module itself or provided externally. Any number of switches may be provided, or none at all. The inputs A and B have two states, logic "0" and logic "P". Those of ordinary skill in the art will recognize that, instead of mechanical switches, the switch signals could be implemented by special touch sensors, operated by charge integrators similar to units 44 which feed into threshold comparators to form digital signals.

Detailed Description Text (192):
Tap unit 280, zigzag unit 282, and push unit 284 examine the sequence of (X,Y,Z)

samples to look for various types of gestures. The outputs of all these units, plus the switch signals, are combined in button control unit 286 to produce the actual button-press signals sent to the host. In the illustrative example disclosed herein, the touchpad simulates a three-button (Left, Middle, Right) pointing device. The system of FIG. 14 could clearly be extended to support other gestures than those described here, or to support fewer gestures in the interest of simplicity.

Detailed Description Text (200):
According to the presently preferred embodiment of the invention, motion events are sent to the host as usual, and also recorded in a register or queue. When the tap gesture is recognized, a corresponding negative amount of motion is quickly replayed in order to "undo" the already-reported motion and to restore the original cursor position as of the moment the finger's presence was first detected. The motion during the stroke may have been sent to the host in the form of a sequence of several packets. For greatest precision, this sequence can be saved and replayed in reverse. However, if the host's motion processing is linear, it will suffice to accumulate the total amount of motion during the stroke and send a compensating motion in a single packet. Since the "acceleration" feature of a typical mouse driver activates only at high speeds, this assumption of linearity is usually safe in this context.

Detailed Description Text (201):
The inputs considered by tap unit 280 are CurPos, the current (X,Y) finger position from the arithmetic unit; Z, the current pressure value; and CurTime, the current time in some suitable units of time (such as milliseconds or number of samples processed).

Detailed Description Text (202):
There are nine state variables used in tap unit 280. TapState is NONE if there is no gesture in progress, TAP if there is a tap or drag gesture in progress, and LOCKED if there is a locking drag or drag extension in progress. TapOkay is TRUE if a high enough Z value has been seen in the current stroke for the stroke to qualify as a tap. DownPos is the (X,Y) position at which the finger last touched down on the pad. DownTime is the time at which the finger last touched down. UpPos and UpTime record the position and time at which the finger last lifted from the pad. TapButton is one of LEFT, MIDDLE, or RIGHT, identifying whether the current gesture is simulating an action on the left, middle, or right virtual mouse button, respectively. Suppress is TRUE if the virtual buttons are being suppressed for a double click. Finally, Out represents the output of the tap unit, and is one of NONE, LEFT, MIDDLE, or RIGHT.

Detailed Description Text (204):
FIG. 15a shows the timing of a basic tap gesture. First, a successful tap is shown, followed by a finger stroke which is too long to qualify as a tap. In the first stroke, the finger is down for time "t1", which is less than TapTime. Also (not shown on FIG. 15a) the (X, Y) motion during time "t1" is less than TapRadius. Finally, the Z signal exceeds threshold Ztap for at least some part of the stroke. Thus, the stroke qualifies as a tap. The Out signal (the lower trace of FIG. 15a) becomes true for a certain amount of time "t2", then becomes false. As will be discussed later, the amount of time "t2" is equal to DragTime. In the device described in the flowcharts to follow, the TapState variable will equal TAP for the entire interval "t2". As presently preferred, TapTime is about 400 msec, TapRadius is about 2% of the width of the sensor pad, and Ztap is slightly larger than Zthresh, whose value is adjustable by the user.

Detailed Description Text (211):
Accordingly, the drag gesture may actually represent two different gestures. A true drag, where the cursor is moved around while the virtual button is being held down, and a press, where the cursor remains stationary while the virtual button is being held down. The drag extension feature is only desired for a true drag. There are several ways to distinguish between a true drag and a press. A true drag can be identified if the finger's speed of motion prior to lift-off is above a small threshold. A press can be identified if the finger was stationary through the entire gesture, possibly ignoring small, inconsequential movements, or just at the time of finger lift-off. In the preferred embodiment of the drag extension gesture of the present invention the distinction between -a true drag and a press is identified by the finger speed at lift-off being above a specified threshold. The finger speed at lift-off is obtained as the output of a running average filter. If the speed is below

the specified threshold, the drag ends rather than being, extended. In an alternative embodiment, the distinction between a true drag and a press may be identified by the position of the finger at lift-off. If the finger is within a selected distance from the edge of the pad at lift-off a true drag is identified.

Detailed Description Text (212):
A second potential problem may occur while using drag extension if the user begins a new unrelated finger action during the ExtendTime period. As discussed above, when drag extension is enabled, a drag will continue even though the finger has been lifted from the touch pad if the finger is brought back to the touch pad within the drag timeout. It may be that the user actually wants the drag to end when the finger is lifted, and to begin a new gesture when bringing the finger back down to the touchpad. One way to determine whether the drag gesture is continuing or is being ended and a new finger action begun is to compare the lift-off finger position and the touchdown finger position. Usually, a subsequent stroke of an extended drag would not begin at the spot where the previous stroke had ended. Therefore, if the finger comes down within a specified distance from the lift-off position (within the specified drag timeout), then the drag extension feature allows the drag to continue, otherwise the drag ends immediately. It will be appreciated, however, by those of ordinary skill in the art that the drag extension feature may be implemented, though not preferably, without comparing the finger position at touch down with the finger position at lift-off, and further, that the drag need not end immediately.

Detailed Description Text (221):
On the other hand, a novice tap is seen having a duration for the interval of between 200 msec and 500 msec (in the preferred embodiment, strokes longer than 500 msec would be disqualified as taps). The virtual button click or OUT signal of 200 msec in the interval "t2b" begins after a delay "t2a" of 300 msec, and as a result the user will have a longer DragTime of 500 msec in which to begin a drag gesture. Those of ordinary skill in the art will recognize that the length of the delay may be chosen in several different ways, including as a function of the tap duration. Similarly, the other time-related parameters of gesture recognition such as HopTime and ExtendTime can be adjusted when novice taps are involved. If the finger comes back down to begin a drag gesture before the delayed click has begun (i.e., during the "t2a" interval), then the virtual button click must begin immediately as the finger comes down. Otherwise, if this new finger stroke also turned out to be a tap, the first click of the resulting double-click could be subsumed in the "t2a" interval.

Detailed Description Text (223):
FIG. 15E shows the drag extension gesture. The drag extension begins with a standard drag involving intervals "t14" through "t16". The finger is raised during interval "t17", but because the finger is off the touchpad for a length of time shorter than the drag timeout parameter ExtendTime, the OUT signal remains high. Also (not shown on FIG. 15E) the (X,Y) motion during "t17" is greater than DragExtendRadius and the smoothed average finger speed at the time of lift-off from the pad at the beginning of interval "t17" is greater than DragExtendSpeed. The figure shows the finger lifted for a second interval "t18". Since the period of time in which the finger is lifted from the touchpad during interval "118" is greater than ExtendTime, the OUT signal goes low a period of time equal to ExtendTime after the finger is lifted from the pad. It may be preferable to adjust ExtendTime for novice or expert users, as described previously for DragTime.

Detailed Description Text (225):
Other gestures may be used to simulate a multi-button mouse. In one such approach, the basic gestures are augmented by a "hop" gesture, in which the finger is lifted from its resting place in one location on the pad and tapped a substantial distance away from the resting place. If the distance is sufficiently great (HopDistance, typically a fraction of the width of the sensor pad; presently preferred to be about 25%) and the duration between the lift and the subsequent tap is less than a suitable threshold (HopTime, typically a fraction of a second; presently preferred to be about 0.5 sec.), then the click or drag gesture begun by the tap is simulated on a different mouse button. This different button may be a fixed "secondary" button, or it may be user-selectable by a control panel or other means, or it may be a function of the direction in which the finger hopped (e.g., to the left vs. to the right). According to a presently preferred embodiment of the invention, the hop gesture is available as

an option which is off by default.

Detailed Description Text (227):
FIG. 15G shows a "hop" gesture. This gesture begins with the finger already on the pad. The finger is then lifted for interval "t24" which is less than HopTime; the finger then comes down for a regular tap "t25". Also, not shown on the figure, during interval "t24" the finger must have moved by at least a certain distance HopDistance away from its previous position. When this occurs, the gesture is processed as a "hop" instead of a regular tap, and the virtual button press "t26" occurs on the right button Out(R) instead of the usual left button Out(L). It is easy to see how the tap "t25" could be followed by further finger actions to form a drag or a double-tap on the right button.

Detailed Description Text (235):
Step 302 determines whether the finger is up or down by comparing Z (pressure) against Zthresh to determine whether a finger is present ("down") or not ("up"). Instead of a simple threshold comparison, two thresholds may be used to provide hysteresis as is well-known in the art. Hysteresis is not shown in FIG. 17a, but similar hysteresis will be illustrated later in FIG. 20 for the "push" gesture.

Detailed Description Text (237):
In step 306, a finger-down transition has been detected. This may indicate the beginning of a drag gesture or a successive row in drag extension, etc. For a drag or drag extension gesture, the change in the finger position from the previous finger position on the touchpad during DragTime and ExtendTime, respectively, is checked.

Detailed Description Text (240):
Since the TapState during a drag gesture is TAP and the TapState during a drag extension gesture is LOCKED, step 306 determines the TapState. If the TapState at step 306 is TAP, then step 308 computes the distance between the current position CurPos (the filtered and smoothed X and Y position data) and the saved position of the previous tap, DownPos. If the distance is greater than some threshold DragRadius, then execution proceeds to step 310. Otherwise, it proceeds to step 312. The threshold DragRadius should be some fraction of the width of the pad, preferably larger (more generous) than the TapRadius used in basic tap detection.

Detailed Description Text (242):
If the TapState of step 306 is LOCKED, and DragLock is not enabled, then a drag extension must be in progress. Step 314 computes the distance between the CurPos and the saved ending position of the previous stroke, UpPos. If the distance greater than some threshold DragExtRadius, then execution proceeds to step 312. Otherwise it proceeds to step 316. The threshold DragExtRadius should be some fraction of the width of the pad, as determined by user testing. (Some users may prefer a DragExtRadius of zero, so that step 310 is effectively disabled.)

Detailed Description Text (246):
Step 312 records the position and the time at which the finger touched down.

Detailed Description Text (248):
Step 320, which executes on all samples in which the finger is down, compares Z against the Ztap threshold; step 322 sets TapOkay to TRUE if Z is greater than the Ztap threshold. Thus, when the finger lifts, TapOkay will be TRUE if Z ever exceeded the tap threshold during the brief stroke that is a candidate for a tap gesture.

Detailed Description Text (251):
In step 328, any finger motion which has occurred is retroactively canceled out by quickly replaying to the host a corresponding negative amount of motion from the register or queue in order to "undo" the already-reported motion and to restore the original cursor position as of the moment the finger's presence was first detected. If the motion during the stroke was sent to the host in the form of a sequence of several packets, this sequence can be saved and replayed in reverse. If the host's motion processing is linear, it will suffice to accumulate the total amount of motion during the stroke and send a compensating motion in a single packet. Since the "acceleration" feature of a typical mouse driver activates only at high speeds, this assumption of linearity is usually safe in this context.

Detailed Description Text (252):
Step 330 takes one of several actions based on the current TapState. First, if TapState is NONE (no gestures in progress), execution simply proceeds to step 332. In step 332, the duration of the tapping stroke, CurTime minus DownTime, is computed to distinguish short, expert taps and long, novice taps. For expert taps, execution simply proceeds to step 338 of FIG. 17c. For novice taps, execution proceeds to step 334, which arranges to use a longer value for DragTime for the current gesture. These steps may simply compare the tap duration to a fixed threshold to choose between two fixed DragTime values, or they may use the tap duration to smoothly modulate the DragTime.

Detailed Description Text (277):
In step 368, which executes whenever the finger is lifted and does not qualify as a tap, UpPos is updated to the current position as described above.

Detailed Description Text (290):
The "zigzag" unit 282 of FIG. 14 decodes a two-finger gesture in which one finger remains resting on the pad while another finger taps to one side of the primary finger. In terms of the (X,Y,Z) information produced by the basic device, this gesture will effectively increase the Z value while quickly shifting the X and/or Y value by a significant distance. (When two fingers are on the pad, the apparent position reported is midway between the two fingers.) If such a change is detected, and is followed by a rapid return to the original X, Y, and Z values, then a tap of a second finger is recognized.

Detailed Description Text (295):
The zigzag unit 282 requires the same position, Z, and time inputs as the tap unit 280. It also requires a speed measure S, which is computed as the distance from the previous to the current finger position at any given time. If any filtering or smoothing done on the normal (X,Y) outputs of the arithmetic unit 16 as previously disclosed, it is best to compute the speed S from the unfiltered (X,Y) values.

Detailed Description Text (296):
State variables of the zigzag unit 282 include ZigZ and ZigZ', which record the two most recent values of Z; ZigPos, and ZigPos', which record the two most recent positions; ZigTime, which records the time at which the presence of the second finger was detected; ZigLeft and ZigRight, which are TRUE if a leftward or rightward zigzag has been detected, respectively; and Out, which represents the output of the zigzag unit 282, and is one of LEFT, RIGHT, or NONE.

Detailed Description Text (297):
The zigzag unit 282 uses several parameters. ZigDistance, the minimum distance the finger position can move to qualify for this gesture. ZigMaxTime is the maximum amount of time the second finger can be present to qualify. Szig is the instantaneous finger speed required to begin the detection of the gesture and is determined experimentally, depending on the sample rate, sensor dimensions, and amount of analog filtering in the charge integrators. ZigRadius and ZigLimit specify how close the position and Z values, respectively, must return to their original pre-zigzag values after the second finger is lifted. ZigRadius is comparable to TapRadius, and ZigLimit is about 30% of Zthresh in the presently preferred embodiment.

Detailed Description Text (302):
Step 408 checks for the beginning of a "zig", the first half of the zigzag gesture in which the apparent finger grows and jumps to one side. The speed S of the current sample is compared against the threshold Szig. If S is greater, and ZigZ contains valid data (not the reserved value NONE), then execution proceeds to further validation of the gesture in FIG. 18b.

Detailed Description Text (303):
In step 410, no incipient "zig" has been seen, so ZigPos' is updated to reflect the most recent finger position, and ZigPos is updated to reflect the second-most-recent finger position. If smoothing or filtering is applied to the output of the arithmetic unit 16 of FIGS. 1 and 8, then, unlike the S computation described earlier, ZigPos should be updated from the filtered or smoothed position data. In other words, it

should be updated from the processed position data which is used to update the cursor position on the host.

Detailed Description Text (307):
Step 416 then initializes the ZigLeft and ZigRight flags. These flags will become TRUE if the finger is seen to move significantly far to the left or right, respectively, of its starting position.

Detailed Description Text (309):
After step 418, CurPos reflects the zigged apparent finger position, and ZigPos reflects the position from two samples before the speed passed the Szig threshold. The two-sample history is important because a small amount of motion may have occurred due to the approaching second finger, before the finger touched down and produced the large motion that exceeded Szig. After step 418, ZigPos contains the current position saved at a time before the second finger is likely to have had an effect. Likewise, ZigZ contains the Z value from before the second finger arrived.

Detailed Description Text (310):
Step 420 checks to see if Z has increased substantially beyond the resting Z value ZigZ. In the presently preferred embodiment, Z is compared against a threshold 30% larger than ZigZ. If Z is too small, the "zig" is disqualified and execution returns to step 404.

Detailed Description Text (316):
Step 432 checks if the second finger has lifted from the pad, by comparing Z against a second "zag" threshold somewhat less than the "zig" threshold of step 420. (In the current system, this threshold is roughly 20% larger than ZigZ.) The "zag" threshold is set below the "zig" threshold in order to provide simple hysteresis.

Detailed Description Text (325):
Another gesture which is useful in specialized applications is a "push" gesture, which simply compares the Z (pressure) information against a second Z threshold ZpushDown, considerably higher than the basic finger-detection threshold, and simulates a mouse button action whenever Z exceeds this threshold. This "push" gesture is similar to the way pen-based pointing devices normally operate; however, it too imprecise and too tiring on the finger to use as the primary click or drag gesture. The "push" gesture is most useful in special contexts such as freehand drawing programs.

Detailed Description Text (326):
FIG. 19 is a timing diagram illustrating a "push" gesture. To perform this gesture, the finger is first brought near enough to cause cursor motion without causing a virtual button press. Next, the finger pressure increases past threshold ZpushDown, causing the virtual button to be pressed. Later, the pressure reduces below a threshold ZpushUp, causing the virtual button to be released. If ZpushUp is somewhat lower than ZpushDown, the resulting hysteresis will prevent unwanted oscillation on the virtual button if the finger pressure varies slightly around the "push" threshold.

Detailed Description Text (331):
Step 454 executes if no "push" gesture is in progress. This step checks if a "push" should begin. First, "push" gestures must be enabled by the user. Second, current Z value must be greater than the threshold ZpushDown.

Detailed Description Text (335):
Those of ordinary skill in the art will note that the tap unit 280 is suitable for use with any touchpad that provides (X,Y) and finger-presence information, and push unit 284 is suitable for use with any touchpad that produces Z (pressure) information. Only the zigzag unit 282 depends on special characteristics of the particular touchpad technology disclosed herein, namely the fact that two fingers reliably report an averaged finger position.

Detailed Description Text (336):
Two more algorithms that are not directly part of gesture processing may be used to address minor problems that occur when the user taps on the pad. Specifically, the finger position sometimes shears sharply in one direction just as the finger lifts

away. This is due to natural slippage of the finger during this action, and is aggravated when the finger is held at a shallow angle. A "reverse motion" algorithm can deal with some jumps so far that the TapRadius test fails, reverse motion cannot help.

Detailed Description Text (337):
If Z is seen to be changing rapidly between the current and previous samples (i.e., if the absolute difference between the current and previous Z values is less than some empirically determined threshold), then the time constant of the (X,Y) filtering of the output of the arithmetic unit 16 can be increased. Normally, the old filter value and new quotient are averaged with roughly equal weighting to produce the new filter value. If Z is rapidly changing, the old filter value is instead weighted considerably (e.g., an order of magnitude) more than the new quotient. The result is that any motion occurring during this instant of high Z change is heavily damped.

Detailed Description Text (338):
Often the spurious motion that arises from a finger-lift occurs all in the very last sample before Z decreases below the finger-down threshold Zthresh. Another solution to the problem of spurious finger-lift motion is the "lift-jump suppression" mechanism, which attempts to suppress this final spurious motion event. FIG. 21 shows an illustrative circuit for performing the lift-jump suppression function.

Detailed Description Text (341):
Delay units 480 and 482 record the previous and second-previous values of S, known as S' and S", respectively. Divider 484 computes the quantity one-half of S, denoted S/2. The lift-jump suppression unit looks for a characteristic relationship among the values S, S', S", and S/2 in an attempt to recognize spurious lift-jump events. One practiced in the art will recognize that S" is not valid until the fourth sample of a given finger stroke; thus, the lift-jump suppression unit is disabled for the first three samples of each stroke. The lift-jump suppression unit also employs a parameter Liftjump, a speed threshold which is determined experimentally and is affected by the sample rate and the sensitivity of the sensor pad.

Detailed Description Text (342):
Comparator 486 checks if the speed S is greater than the threshold Liftjump. Comparator 488 checks to see if the previous speed S' is less than Liftjump, and comparator 490 checks if S' is less than S/2. Similarly, comparator 492 checks to see if the second-previous speed S" is less than Liftjump, and comparator 494 checks if S" is less than S/2. If all five conditions are satisfied, AND gate 496 outputs a "suppress motion" signal which suppresses the action of motion unit 18 for this sample. When motion unit 18 is suppressed, its output (.DELTA.X,.DELTA.Y) is not generated for the current sample, and its delay unit 260 is not clocked.

Detailed Description Text (353):
Musical keyboards (synthesizers, electric pianos) require velocity sensitive keys which can be provided by the pressure sensing ability of this sensor. There are also pitch bending controls, and other slide switches that could be replaced with this technology. An even more unique application comprises a musical instrument that creates notes as a function of the position and pressure of the hands and fingers in a very articulate 3-d interface.